SecureMobile: Enhancing Security in Mobile App Development

Depala Rajeswari
A20526535
rdepala@hawk.iit.edu

ABSTRACT

Data security has become increasingly vital in recent years. The evolution of encryption as a solution and fundamental element of data protection protocols. Multiple measures are required to protect the pooled information. The purpose of this research is to examine the efficacy of encrypting data transmissions over networks. Prior to being sent over a network, sensitive information must first be encrypted using a cryptographic method. Second, the receiver has the option of decrypting the data to view it in its original form.

To eliminate compatibility and upgradeability issues, cryptography is the first abstraction that separates specific algorithms from generic cryptographic processes. Cryptography was invented to do this. The primary goal is to strengthen the protection provided by the RSA algorithm. Comparative research on the public key algorithms RSA and enhanced RSA is presented in this dissertation.

INTRODUCTION

Data communication is vital to our lives. Data must be protected from misuse. Encryption and decryption are cryptosystem data transformations. Encryption keys are used to convert plain text into cipher text. The decryption key converts cipher text to plain text, or the original data. To improve the protection mechanism RSA cryptosystem, which was initially made public in the August 1977 issue of Scientific American, was developed by Ron Rivest, Adi Shamir, and Len Adleman. The most common applications for cryptography are to protect users' privacy and verify the authenticity of digital data.

RSA is currently being used in a variety of commercial systems. It is used to ensure the privacy and authenticity of e-mail, it is used to secure remote login sessions, and it is at the core of systems that process electronic credit card payments. Web servers and browsers use it to secure web traffic. In a nutshell, RSA is frequently utilized in contexts where the safety of digital data is a primary concern.

CRYPTOGRAPHY

Cryptography is a field that focuses on protecting communication from malicious individuals. It utilizes important techniques to ensure the confidentiality, integrity, authentication, and non-repudiation of data in various applications. The Advanced Encryption Standard (AES) is a widely recognized symmetric encryption method that is considered one of the most important cryptographic algorithms. It was officially approved by the National Institute of Standards and Technology in 2001. The Advanced Encryption Standard (AES) is highly regarded for its resilience and effectiveness in protecting sensitive data during transmission or storage. It achieves this by employing data blocks with key sizes of 128, 192, or 256 bits, making it a popular choice for ensuring the security of confidential information. Substitution-permutation networks are employed in its implementation to guarantee strong encryption and decryption.

Another important algorithm in cryptography is the Rivest-Shamir-Adleman (RSA) asymmetric encryption method. RSA, which was coined in honor of its inventors, makes use of the unique mathematical properties of prime numbers. The algorithm functions as a public-key system, employing a set of public and private keys for the purposes of encryption and decryption. RSA is widely used to protect the integrity of digital signatures, key exchange protocols, and communication lines. It's crucial because it allows for safe data transmission and authentication.

Elliptic Curve Cryptography (ECC) revolutionizes asymmetric cryptography. The mathematics of elliptic curves over finite fields for key generation and encryption make it efficient and secure. ECC provides equivalent security with shorter key lengths than RSA, making it ideal for resource-constrained environments like mobile and IoT devices.

LITERATURE REVIEW

The RSA public key cryptosystem, originated by Diffie and Hellman, arose from their pioneering research on exponential key exchange. While RSA holds the top position as the most widely used public key algorithm, Diffie-Hellman key exchange is considered the second most popular. Nevertheless, the notion of a comparable system was initially introduced in 1973 by Clifford Cocks, a mathematician employed at the United Kingdom's Government Communications Headquarters (GCHQ) intelligence agency. However, due to the excessively high cost of computers at that time, the system was not put into action or made known until 1998. In 1976, professors Ronald Rivest, Adi Shamir, and Leonard Adleman from MIT embarked on a cryptographic undertaking. Rivest and Shamir, both computer scientists, collaborated with Adleman, an expert in number theory. After several iterations, they successfully developed the RSA algorithm, which remains resistant to decryption and is a fundamental element in modern cryptography (Ohya & Volovich, 2011).

The RSA core has proven to be resistant to attacks by highly skilled cryptographers. Although no formal mathematical proof exists, the algorithm's ability to withstand challenges provides a strong sense of security. There have been attempts to weaken its outer parts, according to Dan Boney, a computer science professor at Stanford University, but no one has figured out how to access its central part.

TOOLS USED

Python's standard GUI library is tkinter. It helps programmers build interactive applications by providing classes and functions for creating windows, dialogs, buttons, and other GUI components.

The Python cryptography library provides cryptographic algorithms and components with an easy-to-use interface. Importing hazmat helps manage cryptographic operations.

Asymmetric means uneven. RSA encryption and decryption classes and functions are in this submodule.

Binary data like ciphertext is converted to ASCII text using base64 to encode and decode base64-encoded data.

Os module functionality enables communication with the operating system.

ENCRYPTING

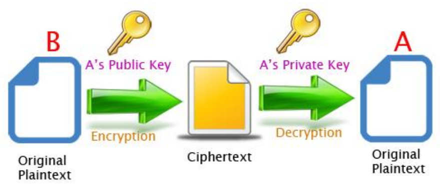
To ensure the confidentiality and integrity of the transmitted message, the code describes an encryption procedure that must be followed. The input message is first encoded into a string of bytes using the UTF-8 encoding scheme. This is an essential step in transforming the text from a human-readable format to one suitable for use in cryptographic operations.

```
def encrypt(self, message, public_key):
        ciphertext = public_key.encrypt(
            message.encode('utf-8'),
            padding.OAEP(
                mgf=padding.MGF1(algorithm=has
hes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )
        return b64encode(ciphertext).decode('utf-
8')
```

Then, the RSA algorithm is used alongside OAEP (Optimal Asymmetric Encryption Padding) to carry out the meat of the encryption. Using a mask generation function (MGF1) and a hash function, the OAEP padding scheme improves RSA's security and has seen widespread adoption as a result.

In this implementation, SHA256 is employed throughout the encryption process, including the mask generation function. Among the available hashing algorithms, SHA256 is widely considered to be one of the safest options due to its inclusion in the SHA-2 family.

For some security systems, the mask generation function (MGF1) is mission-critical. It helps ensure that patterns in the plaintext do not translate into predictable patterns in the ciphertext, adding an extra layer of security to the encryption process.



The ciphertext that is produced after encryption is then base64 encoded. When binary data is encoded with Base64, it can be transmitted across a wide variety of platforms and protocols. Base64 encoding becomes crucial when dealing with text-based communication channels because it ensures a consistent and human-readable representation of the encrypted data.

In summary, the encryption method combines UTF-8 encoding, RSA with OAEP padding using SHA256, and base64 encoding to provide a secure and standardized approach for protecting the confidentiality of messages during transmission. This algorithmic approach is compliant with current cryptographic system standards and is an integral part of a larger encryption scheme developed to safeguard sensitive data in transit.

DECRYPTING

During the decryption process, the algorithm employs a highly secure method by utilizing the RSA private key to decode the ciphertext from its base64 representation. The private key can be imported from a predetermined PEM file or provided as an arbitrary key argument. The choice of Optimal Asymmetric Encryption Padding (OAEP) as the padding scheme is due to its extensive adoption and its ability to provide security against vulnerabilities found in simpler padding schemes. The utilization of SHA-256, a cryptographic hash function, in both the mask generation function (MGF1) and encryption algorithm of OAEP enhances the overall security of the system.

The OAEP padding scheme plays a crucial role in thwarting specific cryptographic attacks, such as the padding oracle attack, by incorporating randomness into the encryption process. This feature enhances the algorithm's security against deliberate attacks on specific ciphertexts by ensuring that any modification to the ciphertext will result in a distinct encrypted message.

```
def decrypt(ciphertext, private_key_path,
custom_private_key=None):
    key_to_use = custom_private_key

    if custom_private_key is None:
        # Load private key from PEM file
        with open(private_key_path, 'rb') as key_file:
            key_to_use =
serialization.load_pem_private_key(
                key_file.read(),
                password=None,  # You might need to
provide a password if your key is encrypted
                backend=default_backend()
            )

    decrypted_message = key_to_use.decrypt(
        b64decode(ciphertext),
        padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA256()
),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted_message.decode('utf-8')
```

The RSA private key is employed for decryption after the decoding of the ciphertext and the application of OAEP padding. Ultimately, the deciphered message is yielded as a string encoded in

UTF-8 format, enabling it to be comprehended by individuals.

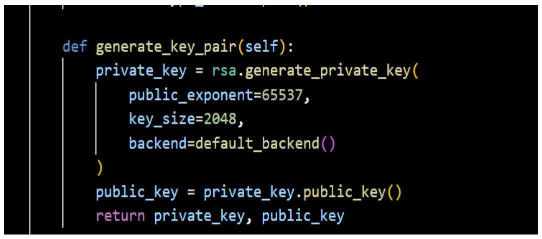
This approach adheres to cybersecurity best practices by assigning equal significance to both the confidentiality of the decryption process and the integrity of the retrieved message. The algorithm's commitment to rigorous security measures in the domain of data protection and confidentiality is emphasized by its utilization of advanced cryptographic techniques such as OAEP with SHA-256, in addition to the conventional RSA decryption steps.

The decryption process ensures both the integrity and authenticity of the recovered message. The algorithm's strict adherence to the standard RSA decryption steps, combined with the utilization of advanced cryptographic techniques such as OAEP with SHA-256, highlights its dedication to strong security measures in the field of data protection and confidentiality.

WORKING OF THE MODEL

Implementing a GUI with the Tkinter library and cryptography with the cryptography library. Bob, our fictitious user, can enter a message into the application's graphical user interface, encrypt it with RSA and the OAEP padding scheme, and then save the encrypted message and his private key to a file.

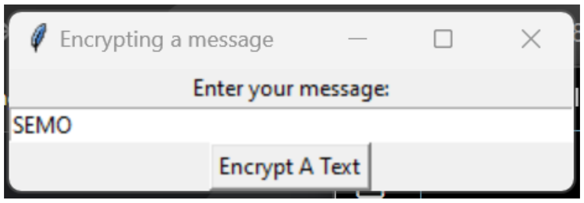
The Encrypting Message class serves as the fundamental element comprising the structure of the application. A set of RSA keys is generated during class initiation; these keys comprise a private key denoted as bob_private_key and a corresponding public key called bob_public_key. The graphical user interface (GUI), which is built upon the Tkinter framework, comprises an entry widget and a label that request the user to submit a message.



```
def generate_key_pair(self):
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key
```

The encrypt method is responsible for carrying out the encryption process. The system takes the user's input, transforms it into UTF-8 encoding, then safeguards it by encrypting it using the RSA public key and the OAEP padding technique.

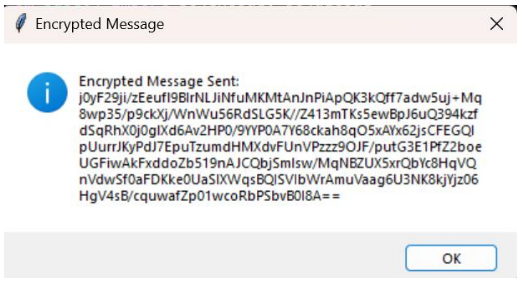
Afterwards, the resulting ciphertext is encoded in base64 format to make it easier to store and display.



Enter your message:
SEMO

The primary execution block of the application creates an instance of the Encrypting Message class, initializes the Tkinter graphical user interface, and starts the event loop.

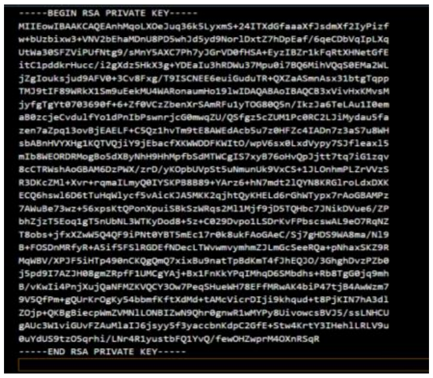
Upon receiving the user's message and activating the encryption button, the encryption procedure is initiated. Upon completion of the procedure, Tkinter's messagebox.showinfo function is utilized to display an information dialog box. This dialog box notifies the user that the communication has been effectively encrypted and saved.



```
Encrypted Message Sent:
j0yF29ji/zEeufI9BlrNLJiNfuMKMtAnJnPiApQK3kQff7adw5uj+Mq
8wp35/p9ckXj/WnWu56RdSLG5K//Z413mTKs5ewBpJ6uQ394kzf
dSqRhX0j0gIXd6Av2HP0/9YYP0A7Y68ckah8qO5xAYx62jsCFEGQI
pUurrJKyPdJ7EpuTzumdHMXdvFUnVPzzz9OJF/putG3E1PfZ2boe
UGFiwAkFxddoZb519nAJCQbjSmIsw/MqNBZUX5xrQbYc8HqVQ
nVdwSf0aFDKke0UaSIXWqsBQISVIbWrAmuVaag6U3NK8kjYjz06
HgV4sB/cquwafZp01wcoRbPSbvB0I8A==
```

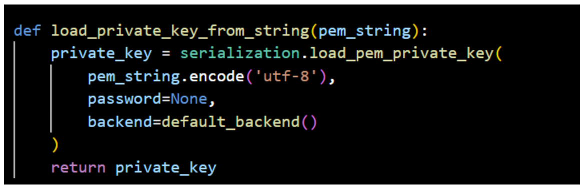
Providing a private key, which may be read from a file or entered as a custom key using the GUI, simplifies the decryption process.

The load_private_key_from_string function was specifically developed to convert a PEM-formatted string that represents an RSA private key into its corresponding cryptographic object. Typically, private key information is stored in a string structured as



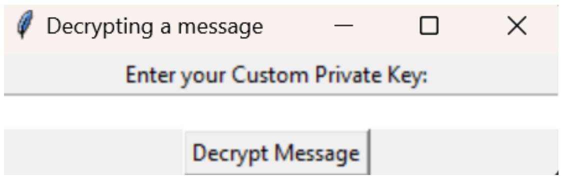
```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAnhMqoLXOeJuq36k5LyxmS+24ITXdGfaaaXfJsdmXf2IyPizf
w+bUzbixw3+VNV2bEhaMDnU8PD5whJd5yd9NorlDxtZ7hDpEaf/6qeCDbVqIpLXq
UtWa30SFZViPUfNtg9/sMnY5AXC7Ph7yJGrVD0fHSA+EyzIBZr1kFqRtXHNetGfE
itC1pddkrHucc/i2gXdz5HkX3g+YDEaIu3hRDWu37Mpu0i7BQ6MihVQqS0EMa2WL
jZgIouksjud9AFV0+3Cv8Fxg/T9ISCNEE6euiGuduTR+QXZaASmnAsx31btgTqpp
TMJ9tIF89WRkX1Sm9uEekMU4WARonaumHo19lwIDAQABAoIBAQCB3xVivHxKMvsM
jyfgTgYt0703690f+6+Zf0VCzZbenXrSAmRFu1yTOG80Q5n/IkzJa6TeLAu1I0em
aB0zcjeCvdulfYo1dPnIbPswnrjcG0mwqZU/QSfgz5cZUM1Pc0RC2LJiMydau5fa
zen7aZpq13ovBjEAELF+C5Qz1hvTm9tE8AWEdAcb5u7z0HFZc4IADn7z3aS7u8WH
sbABnHVYXHg1KQTVQjiY9jEbacfXKWWDDFKWItO/wpV6sx0LxdVypy7SJfleaxl5
mIb8WEORDRMogBo5dXByNhH9HhMpfbSdMTWCgIS7xyB76oHvQpJjtt7tq7iG1zqv
8cCTRWshAoGBAM6DzPWX/zrD/yKOpbUVpSt5uNmunUk9VxCS+1JLOnhmPLZrVVzS
R3DKcZMl+Xvr+rqmaILmyQ0IYSKPB8B89+YArz6+hN7mdt2lQYN8KRGlroLdxDXK
ECQ6hswl6D6tTuHqWlycf5vAicKJA5MKK2qjhtQyKHELd6rGhWTypx7rAoGBAMPz
7AWuBe73wz+56xpsKtQPonXpuiSBkSzWRqs2Ml1Mjf9jD5TQHbc7JNikDVue6/ZP
bhZjzT5Eoq1gT5nUbNL3WTKyDod8+5z+C029Dvpo1LSDrKvFPbscswAL9eO7RqNZ
T8obs+jfxXZwW5Q4QF9iPNt0YBT5mEc17r0k8ukFAoGAeC/Sj7gHDS9WA8ma/Nl9
B+FOSDnMRfyR+A5if5FSlRGDEfNDecLTWvwmvymhmZJLmGcSeeRQa+pNhaxSKZ9R
MqWBV/XPJF5iHTp490nCKQgQmQ7xixBu9natTpBdKmT4fJhEQJO/3GhghDvzPZb0
j5pd9I7AZJH08gmZRpfF1UMCgYAj+Bx1FnKkYPqIMhqD6SMbdhs+Rb8TgG0jq9mh
B/vKwIi4PnjXujQaNFMZKVQCY3Ow7PeqSHueWH78EFfMRwAK4biP47tjB4AwWzm7
9V5QfPm+gQUrKrOgKy54bbmfKftXdMd+tAMcVicrDIji9khqud+t8PjKIN7hA3dl
ZOjp+QKBgBiecpWmZVMNlLONBIZwN9Qhr0gnwR1wMYPy8UivowcsBVJ5/ssLNHCU
gAUc3W1viGUvFZAuMlaIJ6jsyy5f3yaccbnKdpC2GfE+Stw4KrtY3IHehlLRLV9u
0uYdUS9tzO5qrhi/LNr4R1yustbFQ1YvQ/fewOHZwprM4OXnRSqR
-----END RSA PRIVATE KEY-----
```

PEM, and this method converts that text into binary using UTF-8 encoding. After that, it employs the crypto library's serialization function. The private key object is deserialized and constructed using the load_pem_private_key function. Users are provided with a quick and easy way to load an RSA private key from a string thanks to the returned private key object.



```
def load_private_key_from_string(pem_string):
    private_key = serialization.load_pem_private_key(
        pem_string.encode('utf-8'),
        password=None,
        backend=default_backend()
    )
    return private_key
```

The GUI, built using Tkinter, incorporates an entry widget where the user may input their personalized private key. It also contains a label that offers instructions and a button called "Decrypt Message" that initiates the decryption procedure. The graphical user interface (GUI) is uncomplicated and intuitive, facilitating effortless user interaction with the program.



The primary execution block initializes the Tkinter graphical user interface (GUI) window, assigns a title to it, and begins the event loop. When the user interacts with the program by inputting a certain private key and pressing the "Decrypt Message" button, the application proceeds to decrypt the message and presents it in a Tkinter messagebox. The code terminates with a print statement indicating the end of the decryption operation.

To summarize, this program demonstrates a realistic implementation of RSA decryption, allowing the user to utilize a bespoke private key. This emphasizes the need of secure management of private keys in cryptographic operations, illustrating how users may utilize their private key to decode communications in a secure manner. The GUI's simplicity ensures that users who need a direct decryption tool may easily use it, while the cryptographic procedures implemented comply to the highest standards for securely managing data.